Research on Sudoku Puzzles Based on Metaheuristics Algorithm

Xiuqin Deng [1], Junhao Li [2], Guangqing Li [3]

School of Applied Mathematics, Guangdong University of Technology, Guangzhou510006, Guangdong, China

*[1]xiuqindeng@163.com; [2]name_ljh@hotmail.com; [3]346892319@qq.com

Abstract

In the last few years, there has been explosive growth in the application of meta-heuristics algorithm for solving Sudoku puzzles in computer science. In this paper we present a hybrid genetic algorithm (HGA) that uses a "random" technology to improve the performance of genetic algorithm. We also show the performance of the algorithms for solving Sudoku puzzles with 46 different examples. The experimental results indicate that HGA is able to produce very competitive results with respect to meta-heuristics algorithm at a considerably lower computational cost.

Keywords

Sudoku; Hybrid Genetic Algorithm; Metaheuristics Algorithm; Optimization; Random Search Strategy

Introduction

Sudoku

The Sudoku puzzle has become the passion of many people over the world in the past few years. Sudoku is a Japanese logical game that has recently become hugely popular in Europe and North-America. However, the puzzle was first published in New York by the specialist puzzle publisher Dell Magazines in its magazine Math Puzzles and Logic Problems as early as 1979 (History of Dell Magazines), after which it circled through Japan, where it became popular in 1986. Currently in Japan there are now five Sudoku magazines published every month, with a total circulation of over 600,000 (Pendlebury 2005). Since being introduced into the UK by national newspaper *The Daily Mail*, the puzzle's popularity amongst British people has also rocketed. At the time of writing, multiple instances of the puzzle appear daily in many of the UK's most popular newspapers including *The Independent*, *The Times*, *The Sun*, as well as continuing in the *Daily Mail* (Lewis 2007). In 2005, it has become a worldwide phenomenon.

Sudoku is a 9×9 grid made up of 3×3 subgrids, also called "regions." An example of a Sudoku puzzle is shown in Figure 1. Initially, some cells contain "given" numbers that are not allowed to be changed or moved during a process of solving Sudoku puzzles. The goal is to fill in the rest of the empty cells with numbers from 1 through 9, with each digit appearing only once in each row, column, and subgrid. Sudoku rules are clear and simple. The appealing nature of the puzzle lies in the fact that the completion rules are simple, yet the reasoning required to reach the solution may be difficult.

The number of givens does not determine the difficulty of the puzzle. Instead, there are 15 to 20 factors that are claimed to have an effect on the difficulty rating (Semeniuk 2005). The givens can be symmetrical or nonsymmetrical. In the symmetric case, all the givens are located symmetrically with respect to the centre position.

5						3		
	9		5			4		
		4				7		
	5	1		3	7	2	8	9
3		2		8		6		4
		8		5	2	1	3	7
	3	5				9		
6		9				8	2	3
	8			2	3			6

FIG.1 A SUDOKU PUZZLE,WHERE THE NUMBER OF GIVENS IS 37

Related Work

As mentioned above, Sudoku is a very simple and well-known puzzle that has achieved international

popularity in the recent years. The Sudoku problem could be seen as a good test bench for algorithm design and representation, since it is generally known (Semeniuk 2005). Puzzles have been studied for many years in the Computer Science, Mathematics and Artificial Intelligence fields (Lucas and Kendall 2006, Mantere and Koljonen 2009 , Smith 2007) .

As described in (Michael Mepham 2005), three processes (scanning, marking up, and analysis) are the classical techniques for solving sudokus. The application of these processes is not deterministic so that the real performance of a sudoku solver depends on the approaches in their combination: if one applies these processes blindly, a combinatorial blowup may happen.

Gustavo and Miguel (2007) have presented a case study of how a Sudoku puzzle can be solved with rewriting rules using Maude. The main strength of their approach is the naturalness with which Sudoku are represented in Maude and the ease with which the solving procedure is implemented. But the weakness of their implementation lies in its efficiency. Even with the use of strategies to prune the search tree, their implementation cannot compete with the numerous solvers available in the web （A Sudoku solver ， Sudoku Solver and Sudokulist）.

Because Sudoku is known to be NP-complete problem (Yato and Seta 2003), in recent years researchers have started to apply various metaheuristics algorithm to solve the Sudoku puzzles (Deng and Li 2013, Lewis 2007, Lucas and Kendall 2006, Mantere and Koljonen 2009). Lewis (2007) has presented the application of a metaheuristic technique to the popular Sudoku puzzle and proposed a new method based on simulated annealing algorithm for solving Sudoku. He also introduced the new method for producing Sudoku problem instances (that are not necessarily logic-solvable) and used this together with the proposed SA algorithm to try and discover what types of instances this algorithm is best suited.

Mantere and Koljonen(2009) have compared genetic algorithm/ant colony optimization hybrid (GA/ACO) with ant colony optimization (ACO) and genetic algorithm (GA) (2007), cultural algorithm (CA) (2008) for solving Sudoku. Their study showed that GA/ACO generally outperformed the other algorithms and the analysis of the hybrid method and its parameter settings helped improve the GA and CA methods, too, and their results also improved by 14.2% and 19.4%, respectively. Unfortunately, with difficult Sudoku their method is not working well. Out of 100 solve runs, times that Sudoku was able to solve without reinitializations is less than ten.

Genetic Algorithm (GA) (Duan, Wang and Liu 2007) is a random optimization method based on population. The difficult point for solving Sudoku puzzles by genetic algorithm is how to search the global optimal solution and it doesn't make sense to search the near-optimal solution. In order to enhance the convergence speed for solving Sudoku puzzles based on genetic algorithm, Deng and Li (2013) have proposed a novel hybrid genetic algorithm for solving Sudoku puzzles. In the original GA, the crossover and mutation probability are kept fixed during the optimization. In this paper, a new version of GA with the random crossover and mutation probability and the information exchange pattern of particle swarm optimization (Duan, Wang and Liu 2007), named hybrid genetic algorithm (HGA), is introduced. Furthermore, HGA is compared with the GA, CA, ACO and GA/ACO.

This paper is organized as follows. In the next section we describe our hybrid genetic algorithm (HGA). Afterwards, 46 different benchmark Sudokus were used in the experimental studies and experimental results were given. Finally, we draw some conclusions.

Hybrid Genetic Algorithm

Hybrid Genetic Algorithm (HGA) is described in this section. As mentioned above, the objective of the puzzle is to fill in the empty cells, one number in each, so that each column, row, and subgrid contains the numbers 1 through 9 exactly once. Otherwise, the given numbers must stay in the original positions during the solving process. Designing a fitness function that would aid the GA search is often difficult in combinatorial problems (Mantere and Koljonen 2009). In this paper, the conditions that every 3×3 subgrid contains integers from 1 to 9 and the fixed numbers is guaranteed intrinsically and penalty functions are used to force the other conditions. Thus the fitness function w_i is defined as follows:

$$w_i = \sum_{j=1}^{9} (r_j + c_j) \qquad (1)$$

Where r_j is number of repeated digits on the j row while c_j is number of repeated digits on the j column. In the optimal situation, all numbers appear

in every row and column sets, and fitness function value becomes zero.

An integer coded elitist GA (Mantere and Koljonen 2007) is used in this paper. The size of chromosome is 81 integers, divided into nine sub-blocks of nine numbers that correspond to the 3×3 subgrids, from left to right and from top to bottom. Figure 2 shows the corresponding chromosomes of figure 1, in which zero represents a number ready for filling up, non-zero is the given numbers. If number corresponding to non-zero, location would be changed into 1, and the help chromosome shown in Figure 3 could be achieved. The help chromosome is used for checking fixed positions, if there is a number that is not equal to zero that number cannot be changed during the optimization, only the positions that have zero are free to be changed.

500090004	000500000	300400700	051302008	037080052
289604137	035609080	000000023	900823006	

FIG.2 ENCODING CHROMOSOME CORRESPONDING TO A GIVEN SUDOKU PUZZLE

100010001	000100000	100100100	011101001	011010011
111101111	011101010	000000011	100111001	

FIG.3 HELP CHROMOSOME CORRESPONDING TO A GIVEN SUDOKU PUZZLE

To simplify the notation in crossover operation, the chromosomes in figure 2 is denoted by $x = [x_1, x_2, \cdots x_9]$, in which x_i, $i = 1, \cdots 9$ denotes for nine sub-blocks in figure 2. Uniform crossover operation is applied only between sub-blocks, and the sequences of swap mutations only inside the sub-blocks. Therefore the crossover point cannot be inside a 3×3 subgrid but the gene mutation must be inside a 3×3 subgrid.

The procedure of executing HGA can be described in the following:

Initialization Operation

Under the precondition that every 3×3 subgrid contains integers from 1 to 9 and the fixed numbers was guaranteed intrinsically, a chromosome for a given Sudoku puzzle could be achieved by randomly filling it up. Repeat this procedure to achieve 41 chromosomes. Gm is the maximum generation number. Set the current generation G = 0.

Calculate Fitness and Sort

Calculate fitness of the chromosome according to formula (1) and all of 41 chromosomes are then encoded with numerical data linearly ordered by magnitude against the value of w_i in which the chromosome with smallest value is encoded as 1 while with largest value as 41.

Selection Operation

The 21 chromosomes with encoded numbers in the range of 1-21 are selected to generate the next generation of population. First, the fitness of the chromosome with the encoded number 1 increase by 0.25 before selection conducted, remaining the chromosome into the next generation. Then 20 pairs of chromosomes is generated as follows:

$$i = \begin{cases} 6 & 2 \le k \le 5 \\ 11 & 6 \le k \le 10 \\ 16 & 11 \le k \le 15 \\ 21 & 16 \le k \le 21 \end{cases}$$

$$X_1 = rand\ \text{int}(1,1,[2,i])$$
$$X_2 = rand\ \text{int}(1,1,[2,i])$$

Where i is avariable, k stands for an encoded number of chromosome, $rand\ \text{int}(1,1,[1,i])$ stands for a random integer of 1- i generated. In addition, when X_1 and X_2 as parental types are identical, the selection operator would be repeated until both chromosomes become different.

Crossover Operation

HGA applied multiple point cross-over and crossover point must be 3×3 sub-blocks. The cross-points corresponding a 3×3 sub-block in the same location would exchange randomly between two selected chromosome. Since two selected chromosomes differ from each other, thus their encoded numbers are different. Assume that a larger number of encoded chromosomes are denoted by x_b while a smaller number of encoded chromosomes are denoted by x_s, otherwise, the chromosome with the encoded number 1 is denoted by x_1. The chromosome of the Gth generation population x_i^G consists of nine 3×3 sub-blocks, namely: $x_{i1}^G, x_{i2}^G, \cdots, x_{i9}^G$. Set

$$x_1^G = [x_{11}^G, x_{12}^G, \cdots x_{19}^G],\ x_b^G = [x_{b1}^G, x_{b2}^G, \cdots x_{b19}^G]$$

$x_s^G = [x_{s1}^G, x_{s2}^G, \cdots x_{s9}^G]$, The x_1^G, x_b^G, x_s^G performe perform crossover operation to generate the offspring x_i^{G+1}, namely

$$x_i^{G+1} = (x_1^G \otimes_\lambda x_b^G) \otimes_\mu x_s^G \ , i = 2,3,\cdots,21 \quad (2)$$

Where λ and μ are the numbers of sub-blocks taken from one chromosome which are a random natural number to the range [1-8]. First, λ cross-points (i.e. λ sub-blocks) was produced randomly in x_b^G, namely produced λ sub-blocks $x_{bj_1}^G, x_{bj_2}^G, \cdots x_{bj_\lambda}^G$, in which

$j_1, j_2, \cdots, j_\lambda \in \{1,2,\cdots,9\}$, Similarly, μ cross-points was produced randoly in x_s^G, namely produced μ sub-blocks $x_{sr_1}^G, x_{sr_2}^G, \cdots x_{sr_\mu}^G$, in which

$r_1, r_2, \cdots, r_\mu \in \{1,2,\cdots,9\}$. Assume that

$x_1^G \otimes_\lambda x_b^G$ is denoted by $x_t^G = [x_{t1}^G, x_{t2}^G, \cdots, x_{t9}^G]$, then above crossover operation (2) is concretely expressed as

$$x_{tk}^G = \begin{cases} x_{bj_u}^G & k \in \{j_1, j_2, \cdots, j_\lambda\}, u = 1,2,\cdots,\lambda \\ x_{1k}^G & otherwise \end{cases}$$

$$x_{ik}^{G+1} = \begin{cases} x_{sr_v}^G & k \in \{r_1, r_2, \cdots, r_\mu\}, v = 1,2,\cdots,\mu \\ x_{tk}^G & otherwise \end{cases}$$

Mutation Operation

The mutation probability of HGA is a random number and it changes along with the fitness value of the optimal solution in the current population to maintain the diversity of population gene. When the fitness value of the current population's optimal solution is greater than 10, the population gene is rather rich (as the convergence speed is rather quick when the fitness value is less than or equal to 10, which indicates the rather great probability for the cross operators to search and find new chromosomes), with only one-time mutation for each new chromosome. When the fitness value of the optimal solution in the current population is 7, 8, 9 and 10, the mutation times may increase to 5 to solve the problem that it is impossible to generate new chromosomes when the genes tend to be alike along with the convergence process. When the fitness value of the optimal solution in the current population is less than 6, the genes tend to be alike still more, so it is necessary to increase the mutation probability. But the experiment results indicate that too great mutation probability may bring about vibration (which can generate solutions different with those in the current population, but the fitness value is very great, which is useless to convergence). Therefore, the mutation probability of chromosome with encoded numbers of 1-5 may reasonably increase and with encoded numbers of 6-21 may reasonably decrease, which therefore can control the vibration of population fitness within a certain range as well as can avoid population precocious while introducing more abundant genes. Set *k* stands for an encoded number of chromosome and *i* stands for the number of genes of mutation resulted from each chromosome, fitness(1) represents fitness value of the optimal chromosome in the current population, *rand* int(1,1,[1,5]) is a random integer of 1-5 generated. The mutation operation is expressed as:

$$i = \begin{cases} 1 & fitness\ (1) > 10 \\ rand\ \text{int}\,(1,1,[1,5]) & fitness\ (1) > 6 \\ rand\ \text{int}\,(1,1,[1,8]) & k < 6 \\ rand\ \text{int}\,(1,1,[1,3]) & otherwise \end{cases}$$

On the other hand, the mutation operation is performed as swap of two points inside a 3×3 sub-block to make sure that each 3×3 sub-blocks must contain each integer from 1 to 9 in the whole evolution process. Figure 4 shows a more detailed description.

$G = G + 1$

Repeat steps "*Calculate fitness and sort*" to "*G = G + 1*" as long as the number of generations is smaller than the allowable maximum number G_m and the optimal solution is not obtained.

The key novelty of HGA is that besides crossover probability and mutation probability are random numbers, it also has crossover operator possessing dual effects of self-experience and population experience. The self-experience is originated from two selected parental chromosomes, and the optimal chromosome in the current population represents the population experience. Based on these features, a trade-off between efficiency and reliability can be achieved by above random search strategies.

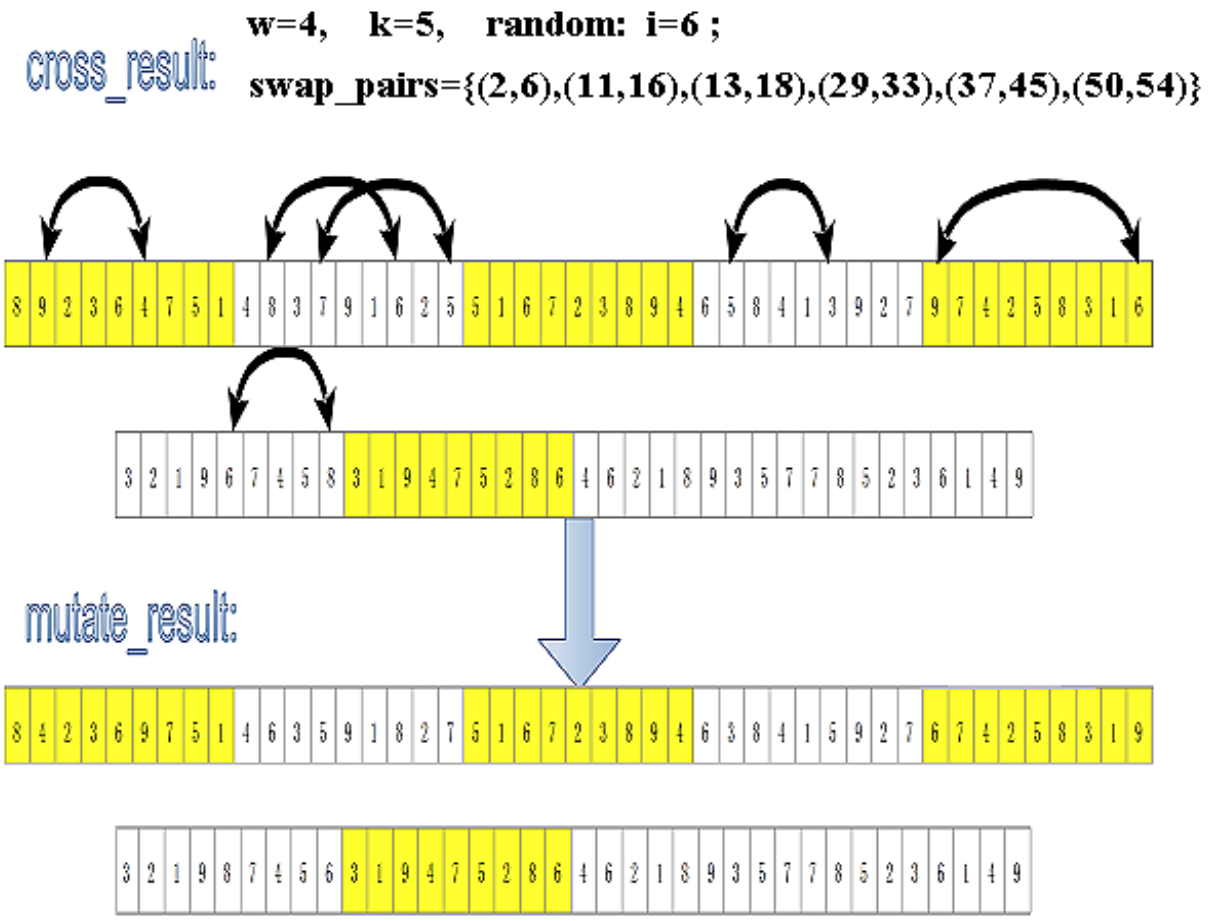

FIG.4 THE DETAILED PROCESS OF MUTATION

Simulation Experiment and Results Analysis

Simulation experiment is conducted to verify the performance and efficiency of HGA. HGA and different meta-heuristics algorithm have been compared. i.e., genetic algorithm (GA), cultural algorithm (CA), ant colony optimization (ACO), and GA/ACO hybrid, by solving 46 different benchmark Sudokus (Mantere and Koljonen: Sudoku project homepage).

15 Sudoku puzzles was taken from the newspaper Helsingin Sanomat (2006), originally rated using 1 to 5 stars. They are denoted by 1, 2, 3, 4, and 5, respectively, in Tables 1 and Tables 2. These had 28 to 33 symmetric givens. We also tested 12 Sudokus taken from newspaper Aamulehti (2006). They were originally rated as: Easy, Challenging, Difficult, and Super difficult. In this paper, they are denoted by E, C, D, and S, respectively. They contain 23 to 36 nonsymmetrical givens. We also have 9 Sudokus taken from Pappocom (2006) rated as, and also subsequently denoted by, Easy, Med., and Hard, and 9 Sudokus we generated with GA (Mantere and Koljonen 2007) rated as GA-Easy, GA-Medium and GA-Hard and subsequently denoted by GA-E, GA-M, and GA-H. The last benchmark Sudoku was the AI Escargot by Inkala (2006). This Sudoku, denoted by AI-E in Tables 1 and Tables 2, has been claimed to be the most difficult Sudoku possible.

With Pentium(R) D CPU 3.11GHz and a memory of 4G as the experiment platform, HGA has been implemented using Matlab7.0. Termination criterion is $w_i = 0$ (i.e. Optimal solution had been found) or iterations are greater than 2000.

We run each the benchmark Sudoku puzzles 100 times and calculate the average generations and success rate for each one, then comes to in Tables 1 and Tables 2. All our methods use self- adapting reinitialization in order to avoid getting stuck in local optima. Table 1 shows how many generations, on average, each method runs, with each benchmark Sudoku, between

reinitializations. At the column of "HGA" of Table 1, "------" means that the algorithm has run more than 2000 generations. It is meaningless to do another more experiments if the algorithm need to run more than 2000 generations to solve a Sudoku puzzles. The data of the other column (GA, CA, ACO, GA/ACO) comes from the table 8 of paper (Mantere and Koljonen 2009). The "avg." of HGA is the average of all the data of each column except those at the locations corresponding to "------". It shows that with easier Sudokus the reinitialization period self-adapts should be shorter than that with difficult Sudokus. On average, GA uses the longest reinitialization intervals, followed by CA, GA/ACO, HGA, and the ACO that uses very short reinitialization periods. This is due to the fact that ACO uses a larger population size than the other methods. With the AI Escargot the reinitialization period was the longest among all the methods. This is due to the fact that with it the near-optimal solution was the most difficult to find. It can be seen from Table1 that HGA outperform GA, CA, and GA/ACO.

Table 2 shows how many times, out of 100 solve runs, each solved Sudoku costs without reinitializations. We get the data of GA, AC, ACO, GA/ACO from table 10 of paper (Mantere and Koljonen 2009), and the data of HGA after running each Sudoku puzzles from (Mantere and Koljonen 2009) 100 times. At Table 2, "0" means that algorithm cannot solve the Sudoku puzzles within 100 trials, in another word, the number of runs, out of 100 runs, that Sudoku puzzles were solved without reinitialization is 0. GA was able to solve each Sudoku at least twice without reinitializations. CA failed to find the solution without reinitializations for only one Sudoku, GA/ACO hybrid for two, and HGA for four. Ant colony optimization needed reinitializations every time for the nine benchmark Sudokus.

HGA needed reinitializations for the four benchmark Sudokus and there are more than GA, CA, and GA/ACO. The reason is the fact that the ination criterion of HGA is optimal solution which had been found or iterations are greater than 2000, but GA, CA, and GA/ACO haven't set the maximum iterations. It is meaningless to do another more experiments if the maximum iterations are greater than 2000. It can be seen from Table 2 that the HGA was the most efficient among the five algorithms. On average, times that every single solved benchmark Sudoku costs without reinitializations within 100 trials are more than any of the other algorithms. HGA can always find the global solution 100% for easy Sudokus. All in all, the success rate of HGA one-time operation is 60%～100% for the Sudoku puzzles with easy and Med , so the success rate is rather high.

From Table 1 and Table 2, although the average generations needed to solve Sudoku puzzles for HGA are more than ACO, the number of runs, out of 100 runs, that Sudoku puzzles were solved without reinitialization is the best. So HGA performs quite well comparing with GA, AC, ACO, GA/ACO (not include difficult level Sudoku puzzles). As a conclusion, HGA is efficient with easy Sudokus, but with difficult Sudokus HGA requiring a lot of trials in order to find the global solution.

Conclusions and Future Work

In this paper, we described in detail random search strategy of HGA and studied performance of HGA. Analysis of the results revealed that HGA outperformed others meta-heuristics algorithm in solving Sudoku puzzles. This might be due to the fact that the cross-point position, crossover probability and mutation probability of HGA is a random number and HGA uses group as a division for priority level in selection operator and under the circumstance of no better chromosome achieved after 4 times of iteration, it should be replaced by a near-optimal chromosome in time, so as to make the searching direction for each iteration searching process is more flexible and the searching scope is wider and population gene is more diverse and the probability of chromosome having more abundant genes for selection is reasonably enlarged and the evolution direction of population can be changed timely. A large number of simulation experiment results showed that HGA is superior to the other meta-heuristics algorithm in terms of success rate and convergence speed.

The HGA worked so well with the Sudoku problem compared to the basic methods that it is planned to test it with other combinatorial optimization problems in the future. For future work we would like to carry out a systematic comparison of the effectiveness of HGA described in this paper with other known approaches for solving Sudoku, especially difficult Sudoku puzzles.

TABLE 1 COMPARISON OF HOW MANY GENERATION IS NEEDED TO SOLVE SUDOKUS WITH DIFFERENT METHODS

	GA			**CA**			**ACO**			**GA/ACO Hybrid**			**HGA**		
	a	**b**	**c**	**a**	**b**	**c**	**a**	**b**	**c**	**a**	**b**	**c**	**a**	**b**	**c**
1	192	181	956	**178**	222	868	28	**21**	54	233	199	742	299	213	731
2	1237	2985	1017	1020	2024	933	91	89	47	884	1516	707	871	898	945
3	1460	3025	1209	1105	1994	949	63	107	76	801	1722	849	853	954	583
4	987	2001	1927	754	1557	1765	68	84	78	839	1095	1085	1033	529	------
5	945	1154	1363	823	946	1071	69	78	92	883	847	1162	------	738	196
E	201	**171**	256	199	213	250	32	57	41	171	291	218	**129**	136	165
C	2825	1976	2892	1749	1335	1895	100	70	96	1505	1152	1346	478	791	561
D	4362	3274	5902	2860	2379	4413	149	95	82	2658	1722	1872	-----	------	------
SD	3887	3872	4637	2789	2803	3412	135	121	113	2803	2407	2573	------	------	-----
Easy	1742	604	374	1157	497	363	109	32	37	667	352	307	882	303	779
Med.	1616	1995	5813	1376	1479	4491	90	103	127	1652	1605	1877	465	------	1143
Hard	4389	1202	3152	3204	1147	2048	145	90	128	3359	1041	2546	------	100	------
GA-E	292	558	261	215	494	220	27	36	22	267	435	184	482	550	561
GA-M	979	370	708	732	330	540	57	44	61	704	409	653	672	225	117
GA-H	1524	1366	3839	1154	1024	2654	101	94	183	1453	1230	3905	-----	------	------
AI_E	**18362**			**11944**			**212**			**5670**			------		
Avg.	**2261.74**			**1642.94**			**83.35**			**1317.35**			**560.71**		

TABLE 2 COMPARISON OF NUMBER OF RUNS , OUT of 100 RUNS , THAT SUDOKUS WERE SOLVED WITHOUT REINITIALIZATION

	GA			**CA**			**ACO**			**GA/ACO Hybrid**			**HGA**		
	a	**b**	**c**	**a**	**b**	**c**	**a**	**b**	**c**	**a**	**b**	**c**	**a**	**b**	**c**
1	58	68	48	56	76	46	70	54	41	48	53	42	100	100	100
2	41	22	31	42	22	22	10	23	30	21	26	27	100	79	90
3	34	15	16	42	19	17	22	6	5	18	11	5	87	88	100
4	9	5	11	7	7	13	4	7	3	7	5	7	69	89	94
5	5	12	10	4	8	9	4	1	0	6	10	0	58	85	71
E	82	83	77	72	86	69	74	54	79	44	67	59	100	100	100
C	33	10	17	27	8	22	12	2	21	31	3	23	96	85	98
D	6	11	22	11	13	25	0	5	14	6	18	15	2	0	47
SD	5	8	5	4	8	3	0	2	1	2	3	5	0	0	0
Easy	34	62	50	39	60	55	58	62	48	37	53	39	100	99	100
Med.	10	2	29	13	2	26	11	1	3	15	4	25	96	100	94
Hard	2	10	4	3	6	1	0	0	2	0	3	4	63	3	2
GA-E	37	52	35	28	52	36	43	39	16	23	29	16	97	99	100
GA-M	14	11	15	11	9	8	7	9	3	11	9	9	90	92	94
GA-H	3	5	4	0	4	1	0	0	0	2	7	2	7	9	6
AI_E	5			3			0			1			8		
Avg.	34.27			33.13			27.3			41.62			45.17		

ACKNOWLEDGMENT

We thank the anonymous referee for valuable comments and suggestions that significantly helped to improve the quality of the paper. The work was supported by the following foundations: Project on the Integration of Industry, Education and Research of Guangdong Province (No. 2012B091100489), Science Technology Project of Guangdong (No.2011B010200042) and Academician Innovation Experiment Project of Guangdong Province (No.1184510144).

REFERENCES

Aamulehti: Sudoku online. Available via WWW: http://www.aamulehti.fi/sudoku/ (cited 11.1.2006).

A Sudoku solver. http://sudoku.sourceforge.net/. (Java)

Deng, X.Q. and Li, Y.D. "A novel hybrid genetic algorithm for solving Sudoku puzzles", Optimization Letters, 7(2), p.241-257, 2013.

Duan, X.D., Wang, C.R., and Liu, X.D. "Particle Swarm Optimization and Application,", p.145-152. Liao Ning University Press, Shen Yang , 2007.

Gustavo, S.G. and Miguel, P. "Solving Sudoku Puzzles with Rewriting Rules", Electronic Notes in Theoretical Computer Science 176, p.79–93, 2007.

Helsingin Sanomat: Sudoku. Available via http://www2.hs.fi/extrat/sudoku/sudoku.html(cited11.1.2006).

History of Dell Magazines. http://www.dellmagazines.com/.

Inkala, A. "AI Sudoku *1002 Vaikeaa Tehtävää"*, Pressmen Oy, 2006.

Lewis, R. "Metaheuristics can solve Sudoku Puzzles", Journal of Heuristics, p.387-401, 2007.

Lucas, S. and Kendall, G. "Evolutionary computation and games", IEEE Computational Intelligence Magazine, 1(1), p.10–8, 2006.

Mantere,T. and Koljonen, J. "Ant Colony Optimization and a Hybrid Genetic Algorithms for Sudoku Solving,"In: 15th International Conference on Soft Computing, Brno, Czech Republic, Mendell 2009, p.41-48 , 2009. The IEEE website. [Online]. Available: http://www.ieee.org/

Mantere, T. and Koljonen, J. "Solving, Rating and Generating Sudoku Puzzles with GA", 2007 IEEE Congress on Evolutionary Computation-CEC2007,Singapore, p1382-1389 , 2007.

Mantere, T. and Koljonen, J. "Solving and analyzing Sudokus with cultural algorithms", 2008 IEEE Congress Computational Intelligence - WCCI2008, 1-6 June, Hong Kong, China, p. 4054-4061, 2008.

Mantere,T. and Koljonen, J.: Sudoku project homepage, Available via. http://lipas.uwasa.fi/~timan/sudoku

Michael Mepham. Solving Sudoku, Daily Telegraph (2005)

Pappocom: Su|do|k*u*. *A*vailable via WWW: http://www.sudoku.com (cited 16.10.2006).

Pendlebury, P. "Can you Sudoku.", In: The Mail on Sunday, London, 8th May2005. An online version is also available at http://www.mailonsunday.co.uk/pages/live/articles/news/news.html?in_article_=348348&in_page_id=1770&in_a_source= (accessed July 2005)

Semeniuk, I. " Stuck on you". NewScientist , 24(31), p.45-47, 2005.

Smith, K. "Dynamic programming and board games: a survey", European Journal of Operational Research, 176(3), p. 299–318, 2007.

Sudoku Solver by Logic v1.4. http://www.sudokusolver.co.uk/. (Javascript)

Sudokulist. http://www.sudokulist.com/. (Step by step solver, with hints at each step.)

Yato, T. and Seta, T. "Complexity and Completeness of Finding Another Solution and Its Application to Puzzles", IEICE Trans. Fundamentals, E86-A (5), p.1052–1060, 2003.